PIT - Práctica 2: Modelos ocultos de Markov (HMM)

Alicia Lozano Díez y Pablo Ramírez Hereza
Estudiante: Gloria del Valle Cano
7 de marzo de 2022

Objetivos

Los objetivos de esta práctica son:

- **Comprobación de las características de un HMM en función de sus parámetros.**
- **Resolución de los tres problemas asociados a un modelo oculto de Markov:**
  - **Cálculo de la verosimilitud de una secuencia de observaciones y un HMM mediante la implementación del algoritmo *Forward*.**
  - **Obtención de la secuencia de estados más probable dada una secuencia de observaciones mediante la implementación del algoritmo de *Viterbi*.**
  - **Entrenamiento de un HMM con *EM*, en concreto el algoritmo *Baum-Welch*.**
- **Utilización del paquete `hmmlearn` de `python` para la implementación de los algoritmos anteriores.**
- **Utilización de HMMs en un caso práctico de clasificación de patrones temporales.**

Materiales - Moodle

Los materiales proporcionados para esta práctica son:

- **Guión (.ipynb) de la práctica**
- **Bases de datos para el ejercicio 3.**
  - **sin_averias.csv**
  - **fallo_componente.csv**
  - **uso_inapropiado.csv**

In [1]:

```
import numpy as np
from hmmlearn import hmm
from matplotlib import pyplot as plt
from sklearn.mixture import GaussianMixture
from scipy.stats import multivariate_normal
```

PARTE 1: Introducción a los Modelos Ocultos de Markov (HMM)

1.1 Definición de un modelo oculto de Markov

Los modelos ocultos de Markov (*Hidden Markov Model* o *HMM*) es un modelo probabilístico generativo, en el cual una secuencia de variables observables X es generada por una secuencia de estados ocultos internos Z. Las transiciones entre estados se asume que siguen la forma de una cadena de Markov de primer orden, de tal forma que el estado en un instante determinado *t* solo depende del estado del modelo en el instante anterior *t-1*.

Un HMM se encuentra definido por:

- **El número de estados del modelo N.**
- **Probabilidades iniciales de ocupación de cada estado π.**
- **Matriz de probabilidades de transición entre estados A.**
- **Distribución de probabilidad de observación de cada estado B.**

a. Dibuje el diagrama de estados correspondiente al HMM definido por los parámetros descritos a continuación:

`Nota:` **Incluir el diagrama en una imagen en la entrega o en el informe de la práctica.**

- **Número de estados, $N = 4$.**
- **Probabilidades iniciales de ocupación de cada estado:**

$$\pi = [0.4\ 0.3\ 0.2\ 0.1]$$

.

- **Matriz de probabilidades de transición:**

$$A = \begin{bmatrix} 0.75 & 0.1 & 0.05 & 0.1 \\ 0.1 & 0.75 & 0.1 & 0.05 \\ 0.05 & 0.1 & 0.75 & 0.1 \\ 0.1 & 0.05 & 0.1 & 0.75 \end{bmatrix}$$

- **Cada estado tiene una distribución de observación (o distribución de emisión) Gaussiana bivariada. Cada Gaussiana se encuentra caracterizada por los siguientes parámetros:**
  - **B_1:**

$$\mu_1 = [-1,\ 0];\ \Sigma_1 = \mathcal{I} * 4$$

.

- **B_2:**

$$\mu_1 = [5,\ -1];\ \Sigma_1 = \mathcal{I}$$

.

- **B_3:**

$$\mu_1 = [4,\ 7.5];\ \Sigma_1 = \begin{bmatrix} 5.0 & -2.0 \\ -2.0 & 3.0 \end{bmatrix}$$

.

- **B_4:**

$$\mu_1 = [-7.5,\ 0];\ \Sigma_1 = \begin{bmatrix} 1.0 & 0.0 \\ 0.0 & 4.0 \end{bmatrix}$$

`Nota` **: Haga uso de la clase** `GaussianHMM` **de** `hmmlearn` **para la implementación de este modelo.**

b.) Inicialice el HMM anterior mediante el uso de la clase `hmmlearn.GaussianHMM`**.**

El paquete `hmmlearn` **[https://hmmlearn.readthedocs.io/en/latest/api.html] es un paquete de software libre desarrollado mediante el uso de scikit-learn, Numpy y matplotlib para el desarrollo e implementacion de los algoritmos destinados al entrenamiento, inferencia y muestreo de diferentes tipos modelos ocultos de Markov (HMM).**

In [2]:

```
# Definición de las variables
# Número de estados
N = 4

# Matriz de Probabilidad de Transicion
A = np.array(
    [
        [0.75, 0.1, 0.05, 0.1],
        [0.1, 0.75, 0.1, 0.05],
        [0.05, 0.1, 0.75, 0.1],
        [0.1, 0.05, 0.1, 0.75],
    ]
)

# Probabilidades iniciales de estado
pi = np.array([0.4, 0.3, 0.2, 0.1])

# TO DO: Utilización DE HMM.gaussianHMM para la definición del modelo
# Fijar la matriz de probabilidades de transición y los estados iniciales.
HMM = hmm.GaussianHMM(n_components=N, covariance_type="full")
HMM.startprob_ = pi
HMM.transmat_ = A

# Vectores de medias
mu = np.array([[-1.0, 0.0], [5.0, -1.0], [4, 7.5], [-7.5, 0.0]])

# Matrices de covarianza
Sigma = []
Sigma.append(np.identity(2) * 4)
Sigma.append(np.identity(2))
Sigma.append(np.array([[5.0, -2], [-2, 3.0]]))
Sigma.append(np.array([[1.0, 0.0], [0.0, 4.0]]))
Sigma = np.array(Sigma)


# TO DO: fijar las medias y las matrices de covarianza de las distribuciones de observación
HMM.means_ = mu
HMM.covars  = Sigma
```

In [3]:

```
HMM.sample(100)
```

Out[3]:

```
(array([[ 3.30945196,  9.20876459],
        [ 2.76891026, 11.13215937],
        [ 9.25642155,  5.34257746],
        [ 4.69835624,  6.15856537],
        [-2.53194584, -1.19007187],
        [ 1.35506459,  0.4164466 ],
        [-3.49993321, -1.70038964],
        [-1.86255988,  2.67180411],
        [-1.41155464, -0.41489011],
        [ 4.71729988, -0.5807938 ],
        [ 5.0211478 , -1.59353484],
        [ 4.87398306, -1.35169753],
        [ 5.7541839 , -1.4927949 ],
        [-6.94998471,  1.68459857],
        [-6.06913792, -0.83688705],
        [-8.1949164 , -2.39437486],
        [-7.4979938 ,  0.60462537],
        [-6.40733151,  0.61029348],
        [-0.66236786,  0.11565749],
```

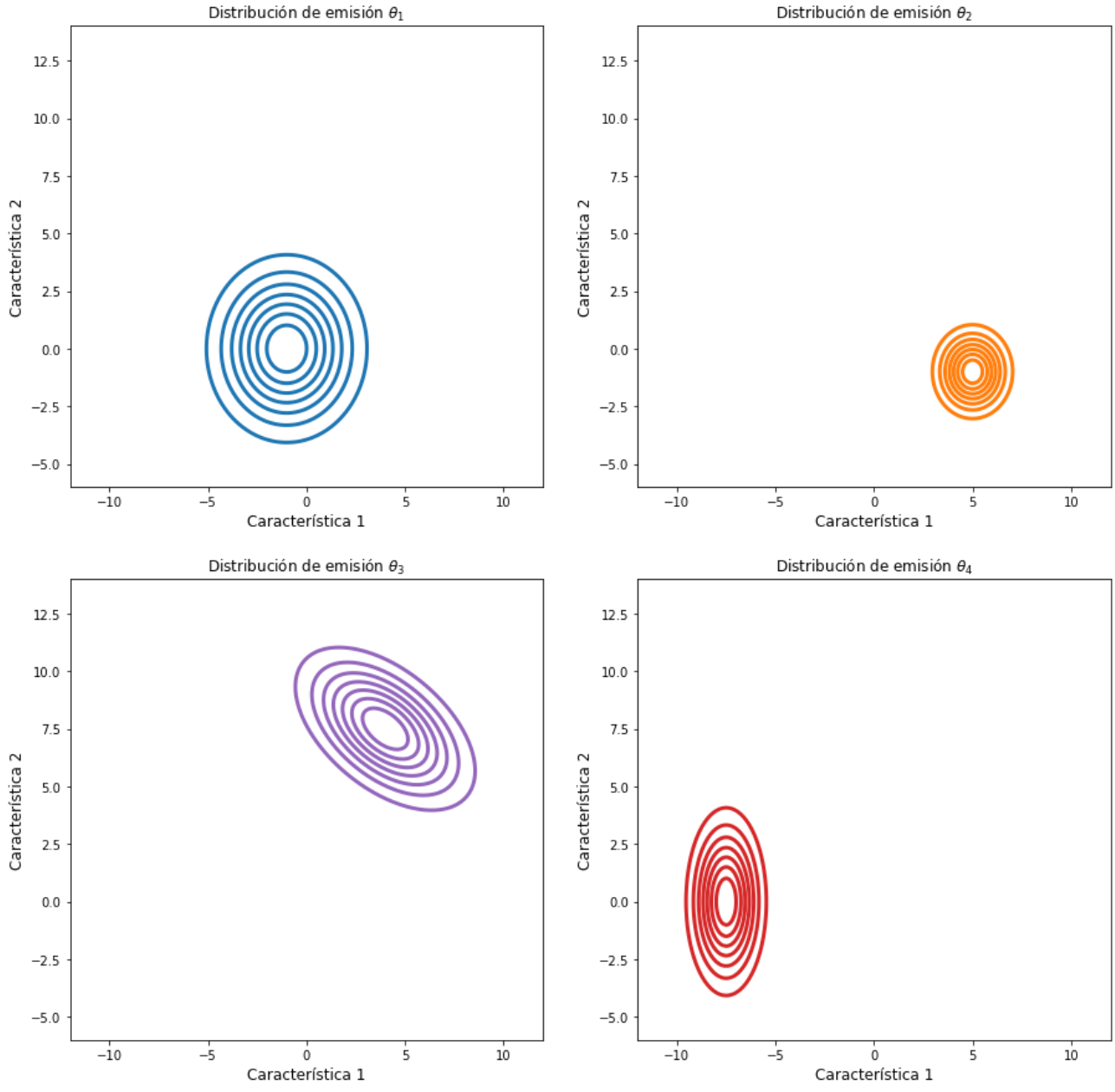
```
        [ 2.5797885 , -1.95675214],
        [-2.48486682, -3.35805127],
        [-7.30572994,  1.01692484],
        [-8.05185157, -0.10533735],
        [-6.79235912,  4.03973945],
        [-6.74645211,  2.6171224 ],
        [-8.32716566,  2.59486214],
        [-6.98584886, -2.16290378],
        [-7.57888786, -1.97106019],
        [-8.78430364, -1.21552094],
        [-3.11270851, -3.09931761],
        [ 5.91823971,  8.67140757],
        [ 6.71374672,  5.22948904],
        [ 3.39483442,  6.87196866],
        [-1.32691892, -1.29371558],
        [ 1.59267319,  1.26712916],
        [-2.29127496, -0.70313085],
        [-0.08907661, -0.87235995],
        [ 0.71751536,  1.97648862],
        [-0.27981785, -4.48646938],
        [-0.97038411,  1.10070516],
        [ 1.78452961,  0.97793133],
        [-2.8718788 , -0.17010198],
        [-0.04905636, -0.32484154],
        [-0.83579933,  3.29278697],
        [-2.39793991, -0.44181003],
        [-0.80084738,  3.43365697],
        [-2.15327962,  3.5244446 ],
        [ 0.7360158 , -3.46199475],
        [-0.41119817,  0.37210886],
        [-2.88452322,  1.08670024],
        [ 4.64248139, -0.4390733 ],
        [ 3.90777008,  0.74334587],
        [ 3.60362897, -1.25664656],
        [ 3.74532921,  9.06873192],
        [ 3.22181452,  8.5830039 ],
        [ 2.6399234 ,  4.83462054],
        [ 0.55578685, 11.6913469 ],
        [ 6.10527552,  4.9916001 ],
        [-1.43709617,  1.35466086],
        [ 5.1240575 , -1.29873184],
        [-1.54218085,  0.87326097],
        [ 5.16477013, -0.15210226],
        [ 3.21524408, -0.81786534],
        [-8.26186263, -2.70696255],
        [-7.76510725,  1.98332206],
        [ 8.13491237,  7.271089  ],
        [ 1.18229343,  8.38680545],
        [ 5.11029951,  7.25066115],
        [-0.55438704,  7.30055661],
        [ 3.8219746 ,  5.61442081],
        [ 4.6867335 ,  7.41837707],
        [ 3.92554082, -1.88174377],
        [ 4.40253766, -1.97371594],
        [ 7.09684684,  4.97720356],
        [ 5.66150502,  6.85244676],
        [ 4.99884032,  6.95288241],
        [ 0.72151303,  7.88874642],
        [ 0.50862722,  8.27810421],
        [-7.98699541, -2.23892792],
        [-6.37031576,  2.28343762],
        [-0.10322072,  0.44604889],
        [-3.87821365,  0.27869875],
        [-2.68549769,  1.80633698],
        [-4.24467904, -1.98698483],
        [ 0.9543563 ,  0.27224047],
        [-6.85334739,  2.59940553],
        [-7.71695984,  2.04923881],
        [ 5.12513949,  5.60357677],
        [ 3.38521303,  8.85789511],
        [ 6.27859284,  7.9219933 ],
        [ 5.47612929,  8.84861769],
        [ 3.20452184,  6.71418353],
        [ 4.1606992 , -0.09288793],
        [ 4.79587928, -0.93131312],
        [ 5.86192689, -2.46106188],
        [ 1.23421279,  8.1280472 ],
        [ 3.95656404,  7.04589671],
        [ 7.53215476,  7.72954875],
        [-5.96265821, -4.15661301],
        [-7.20780243,  0.62248963]]),
 array([2, 2, 2, 2, 0, 0, 0, 0, 0, 1, 1, 1, 1, 3, 3, 3, 3, 3, 0, 0, 0, 3,
        3, 3, 3, 3, 3, 3, 3, 0, 2, 2, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 1, 1, 1, 2, 2, 2, 2, 2, 0, 1, 0, 1, 0, 3, 3, 2,
        2, 2, 2, 2, 2, 1, 1, 2, 2, 2, 2, 2, 3, 3, 0, 0, 0, 0, 0, 3, 3, 2,
        2, 2, 2, 2, 1, 1, 1, 2, 2, 2, 3, 3]))
```

c.) Represente mediante curvas de contorno (función `contour` de pyplot) las distribuciones de probabilidad de observación de cada estado. Para la evaluación de la distribución de probabilidad en el plano de variables utilice la clase `scipy.stats.multivariate_normal`.

In [4]:

```
# Representación de las distribuciones de emisión para cada estado
# Definición de los ejes
x_axis = np.linspace(-12, 12, 300)
y_axis = np.linspace(-6, 14, 300)
```

```
# Generación de matrices con los ejes
_X, _Y = np.meshgrid(x_axis, y_axis)
# Agrupamiento por pares
positions = np.vstack([_X.ravel(), _Y.ravel()]).T

colors = ["tab:blue", "tab:orange", "tab:purple", "tab:red"]


# Generación de la figura
fig, ax = plt.subplots(2, 2, figsize=(15, 15))
ax = ax.ravel()

for n in range(N):

    # TO DO: Evaluación de cada una de las componentes gaussianas en los ejes
    # Utilizar la clase scipy.stats.multivariate_normal
    eval_multivariate_normal = multivariate_normal.pdf(
        positions, HMM.means_[n], HMM.covars_[n]
    )

    prob = np.reshape(eval_multivariate_normal, _X.shape)
    # Representación del contorno
    ax[n].contour(x_axis, y_axis, prob, colors=colors[n], linewidths=3)

    # Etiquetas y título
    ax[n].set_xlabel("Característica 1", fontsize=12)
    ax[n].set_ylabel("Característica 2", fontsize=12)
    ax[n].set_title(r"Distribución de emisión $\theta_{:.0f}$".format(n + 1))

plt.show();
```



1.2 Proceso de generación de una secuencia

Vamos a hacer uso de un dataset sintético generado exclusivamente para su utilización en esta sesión de prácticas. Para la generación de esta base de datos hemos inicializado en el apartado anterior un modelo oculto de markov haciendo uso del paquete `hmmlearn`. Una vez inicializado el modelo, en este apartado vamos a aplicar el proceso de generación de secuencias para extraer muestras del modelo generativo.

a.) Utilice la función `gaussianHMM.sample` para generar una secuencia de 4000 muestras del HMM.

In [5]:

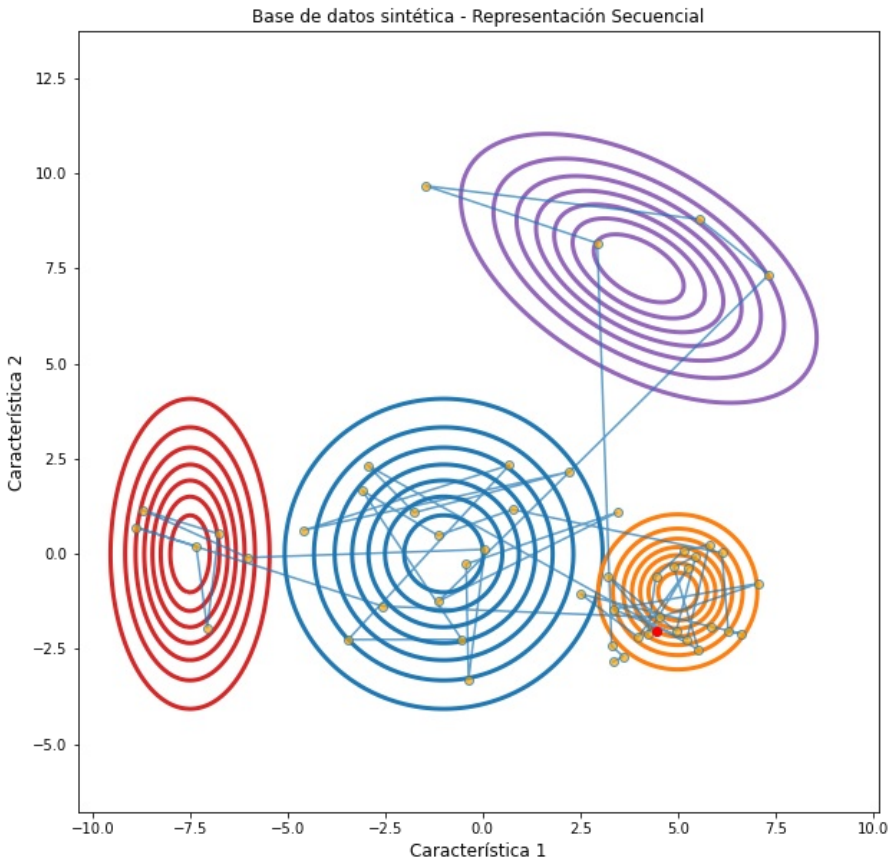
```
# Número de muestras de la secuencia
T = 4000
# TO DO: utilizar gaussianHMM.sample para generar la base de datos.
# X: np.array NxD -> Secuencia de observaciones
# Z: np.array Nx1 -> Estados ocultos
X, Z = HMM.sample(4000)
```

b.) Represente las primeras `T_repr=50` muestras generadas mediante un scatter plot. Adicionalmente, con el objetivo de visualizar el caracter secuencial de los datos, represente cada una de las muestras unidas con líneas.

A partir de la representación, la matriz de probabilidades de transición y los estados ocultos generados, analice el comportamiento del modelo.

In [6]:

```
T_repr = 50

# Representación de los datos mediante un scatter plot
fig, ax = plt.subplots(figsize=(10, 10))

# Representación de los datos mediante un scatter plot
# Para mostrar el desarrollo de la secuencia
ax.plot(
    X[:T_repr, 0], X[:T_repr, 1], "o-", label="observations", mfc="orange", alpha=0.7
)
ax.plot(X[0:1, 0], X[0:1, 1], "or")
plt.xlim((np.min(X[:, 0]), np.max(X[:, 0])))
plt.ylim((np.min(X[:, 1]), np.max(X[:, 1])))
ax.set_xlabel("Característica 1", fontsize=12)
ax.set_ylabel("Característica 2", fontsize=12)
ax.set_title("Base de datos sintética - Representación Secuencial")

for n in range(N):
    #  TO DO: Evaluación de cada una de las componentes gaussianas en los ejes
    # y representación de los contornos
    eval_multivariate_gaussian = multivariate_normal.pdf(
        positions, HMM.means_[n], HMM.covars_[n]
    )

    multivariate_normal.pdf(positions, HMM.means_[n], HMM.covars_[n])
    prob = np.reshape(eval_multivariate_gaussian, _X.shape)
    ax.contour(x axis, y axis, prob, colors=colors[n], linewidths=3)
```



1.3 Comparativa con otros modelos ocutos de Markov.

En este apartado se propone la definición de dos modelos alternativos al generado anteriormente.

a.) Repita los apartados del ejercicio 1.1 y 1.2 con el modelo `HMM2` definido por los parámetros que se representan a continuación:

NOTA: Incluir una imagen representando el diagrama de estados en el informe de la práctica.

- **Número de estados,** $N = 4$.
- **Probabilidades iniciales de estado:**

$$\pi = [0.4\ 0.3\ 0.2\ 0.1]$$

.

- **Matriz de probabilidades de transición:**

$$A = \begin{bmatrix} 0.75 & 0.1 & 0.05 & 0.1 \\ 0.1 & 0.75 & 0.1 & 0.05 \\ 0.05 & 0.1 & 0.75 & 0.1 \\ 0.1 & 0.05 & 0.1 & 0.75 \end{bmatrix}$$

- **Cada estado dispone de un Modelo de Mezclas de Gaussianas (GMM) bivariado como distribuciones de probabilidad de observación. Cada GMM se encuentra caracterizado por:**
  - $\mathbf{B}_1$**:**

$$\pi_1 = [0.3\ 0.3\ 0.4]\,;\ \mu_1 = \begin{bmatrix} -2.0 & 2.0 \\ 0.0 & -1.0 \\ -3.0 & 1.0 \end{bmatrix};\ \Sigma_1 = \mathcal{I}$$

    .
  - $\mathbf{B}_2$**:**

$$\pi_2 = [0.2\ 0.5\ 0.3]\,;\ \mu_2 = \begin{bmatrix} 5.0 & -1.0 \\ 5.0 & 1.0 \\ 6.0 & -1.0 \end{bmatrix};\ \Sigma_2 = \mathcal{I}$$

    .
  - $\mathbf{B}_3$**:**

$$\pi_3 = [0.4\ 0.3\ 0.3]\,;\ \mu_3 = \begin{bmatrix} 3.0 & -5.0 \\ 1.0 & 10.0 \\ 5.0 & 8.0 \end{bmatrix};\ \Sigma_3 = \mathcal{I}$$

    .
  - $\mathbf{B}_4$**:**

$$\pi_1 = [0.5\ 0.3\ 0.2]\,;\ \mu_1 = \begin{bmatrix} -8.0 & -0.0 \\ -8.0 & 2.0 \\ -8.0 & -2.0 \end{bmatrix};\ \Sigma_1 = \mathcal{I}$$

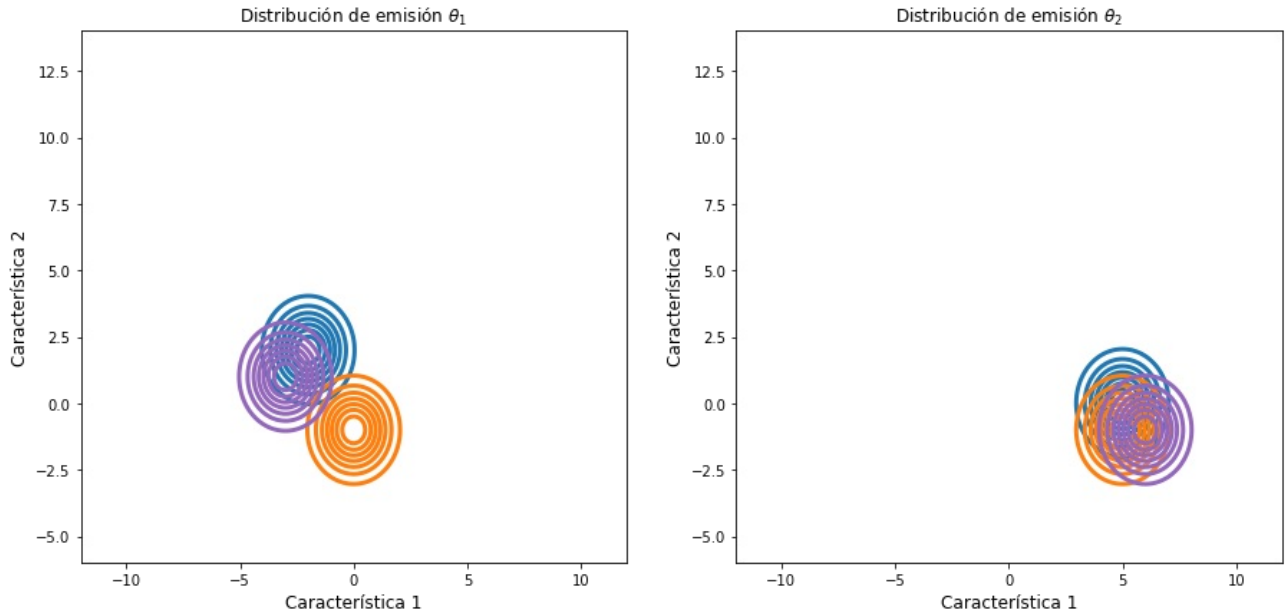
    .

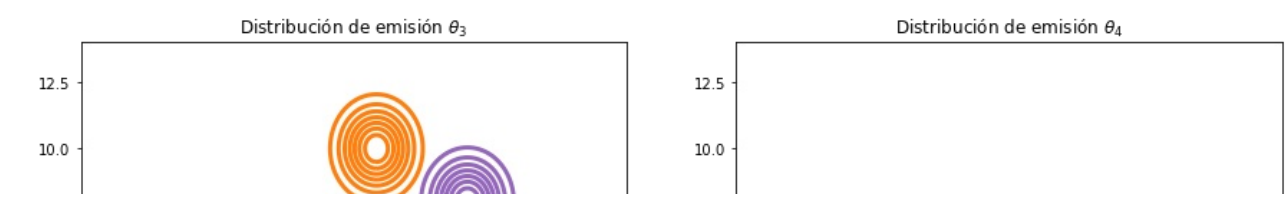
¿Qué tipo de HMM es según su topología?¿En qué difieren los modelos `HMM1` y `HMM2`?

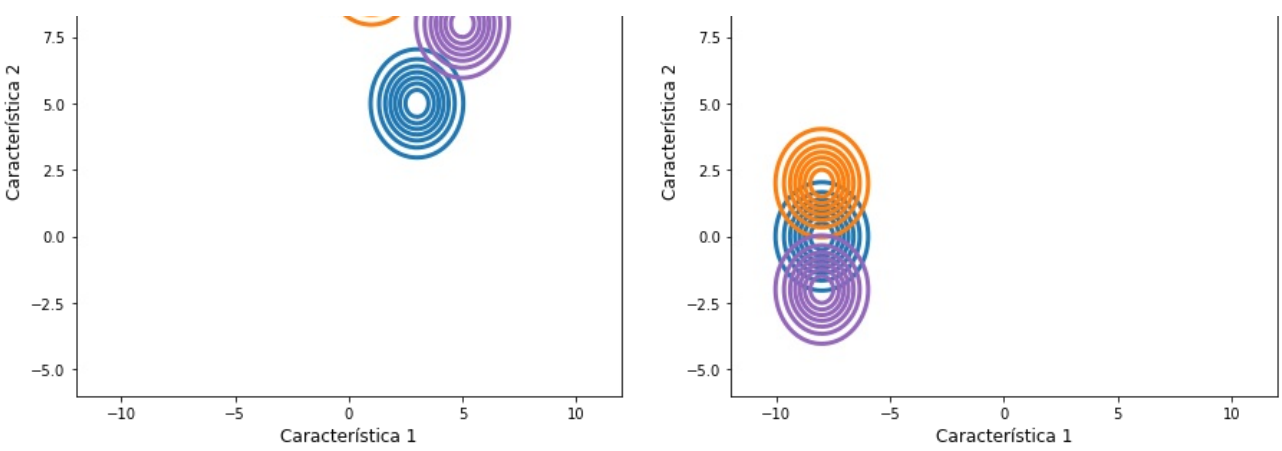
In [7]:

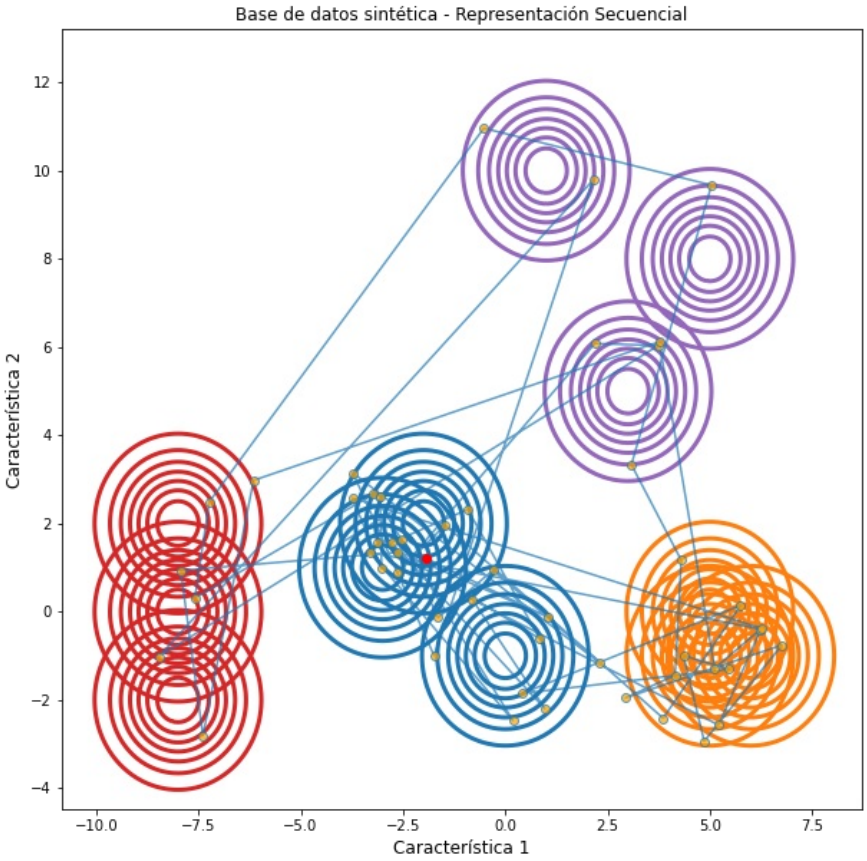
```
# Distribuciones de Probabilidad de emisión: GMM para cada estado
# Numero de componentes gaussianas
n_gaussians = 3

# 1) Pesos de las componentes gaussianas de las GMM
# pesos_{i,j} corresponde al peso de la componente j del estado i ''
weights2 = np.array(
    [[0.3, 0.3, 0.4], [0.2, 0.5, 0.3], [0.4, 0.3, 0.3], [0.5, 0.3, 0.2]]
)

# 2) Vectores de medias de las componentes gaussianas de las GMM
# mu_{i,j} corresponde al vector de medias de la componente j del estado i (vector 2D)
mu2 = np.array(
    [
        [[-2.0, 2.0], [0.0, -1], [-3.0, 1.0]],
        [[5.0, 0.0], [5.0, -1.0], [6.0, -1.0]],
        [[3.0, 5.0], [1.0, 10], [5.0, 8.0]],
        [[-8.0, 0.0], [-8.0, 2], [-8.0, -2.0]],
    ]
)


# 3) Matrices de covarianza de las componentes Gaussianas de las GMM
# sigma_{i,j} corresponde a la matriz de covarianzas de la componente j del estado i (vector 2D)
sigma2 = np.tile(np.identity(2), (4, 3, 1, 1))

# Inicialización del modelo:
# TO DO: Generación de la variable HMM2 de clase GMMHMM
HMM2 = hmm.GMMHMM(n_components=N, n_mix=n_gaussians, covariance_type="full")
HMM2.startprob_ = pi
HMM2.transmat_ = A
HMM2.weights_ = weights2
HMM2.means_ = mu2
HMM2.covars_ = sigma2

######################################################################## 1.1C
# Representación de las distribuciones de emisión para cada estado
```

```
# Definición de los ejes
x_axis = np.linspace(-12, 12, 300)
y_axis = np.linspace(-6, 14, 300)

# Generación de matrices con los ejes
_X, _Y = np.meshgrid(x_axis, y_axis)
# Agrupamiento por pares
positions = np.vstack([_X.ravel(), _Y.ravel()]).T

colors = ["tab:blue", "tab:orange", "tab:purple", "tab:red"]

# Generación de la figura
fig, ax = plt.subplots(2, 2, figsize=(15, 15))

ax = ax.ravel()
for n in range(N):

    for g in range(n_gaussians):
        # TO DO: Evaluación de cada una de las componentes gaussianas en los ejes
        eval_multivariate_gaussian = multivariate_normal.pdf(
            np.dstack((_X, _Y)), mean=HMM2.means_[n][g], cov=HMM2.covars_[n][g]
        )
        prob = np.reshape(eval_multivariate_gaussian, _X.shape)
        # Representación del contorno
        ax[n].contour(x_axis, y_axis, prob, colors=colors[g], linewidths=3)
        # Etiquetas y título
        ax[n].set_xlabel("Característica 1", fontsize=12)
        ax[n].set_ylabel("Característica 2", fontsize=12)
        ax[n].set_title(r"Distribución de emisión $\theta_{:.0f}$".format(n + 1))

plt.show()

########################################################################## 1.2
X2, Z2 = HMM2.sample(T)

T_repr = 50

# Representación de los datos mediante un scatter plot
fig2, ax2 = plt.subplots(figsize=(10, 10))

# Representación de los datos mediante un scatter plot
# Para mostrar el desarrollo de la secuencia
ax2.plot(
    X2[:T_repr, 0], X2[:T_repr, 1], "o-", label="observations", mfc="orange", alpha=0.7
)
ax2.plot(X2[0:1, 0], X2[0:1, 1], "or")
plt.xlim((np.min(X2[:, 0]), np.max(X2[:, 0])))
plt.ylim((np.min(X2[:, 1]), np.max(X2[:, 1])))
ax2.set_xlabel("Característica 1", fontsize=12)
ax2.set_ylabel("Característica 2", fontsize=12)
ax2.set_title("Base de datos sintética - Representación Secuencial")


for n in range(N):
    for g in range(n_gaussians):
        # TO DO: Evaluación de cada una de las componentes gaussianas en los ejes
        eval_multivariate_gaussian = multivariate_normal.pdf(
            np.dstack((_X, _Y)), mean=HMM2.means_[n][g], cov=HMM2.covars_[n][g]
        )
        prob = np.reshape(eval_multivariate_gaussian, _X.shape)
        ax2.contour(x axis, y axis, prob, colors=colors[n], linewidths=3)
```









b) Repetir los apartados del ejercicio 1.1 y 1.2 utilizando un modelo `HMM3` que se diferencia del modelo `HMM1` en la matriz de probabilidades de transición y las probabilidades iniciales de estado:

$$A = \begin{bmatrix} 0.7 & 0.15 & 0.15 & 0 \\ 0 & 0.8 & 0.15 & 0.05 \\ 0 & 0 & 0.9 & 0.1 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\pi = [0.9\ 0.1\ 0.0\ 0.0]$$

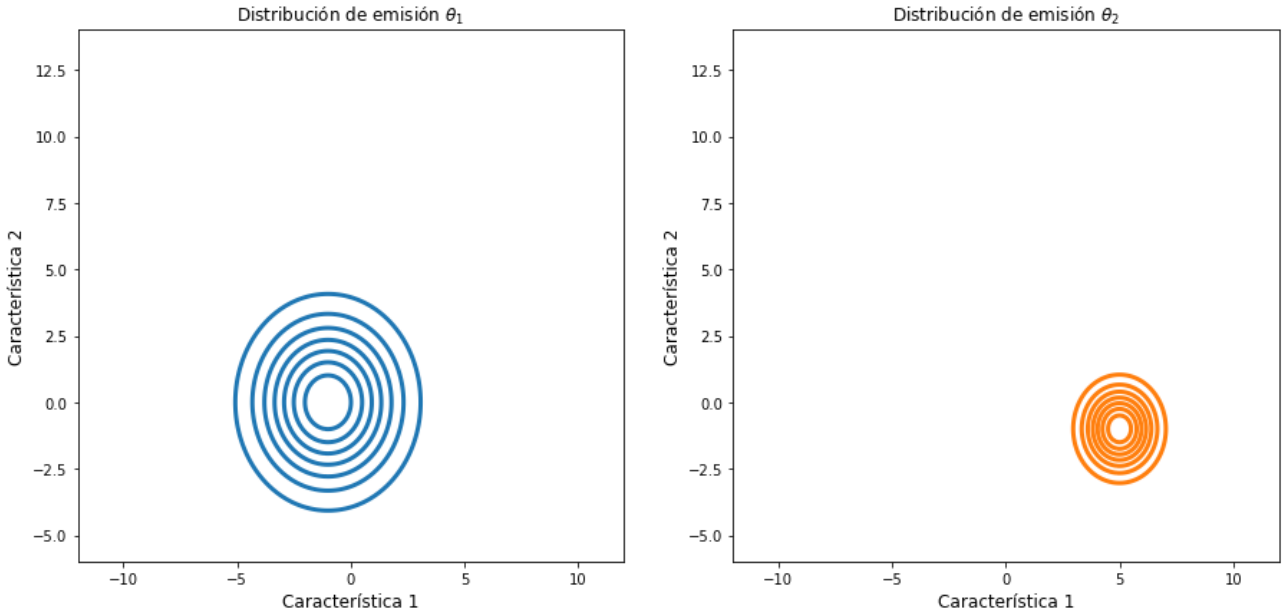
`Nota` **: Las secuencias X y Z generadas tras el muestreo de `HMM3` se guardarán en las variables X3 y Z3, para no sobreescribir las secuencias ya generadas.**

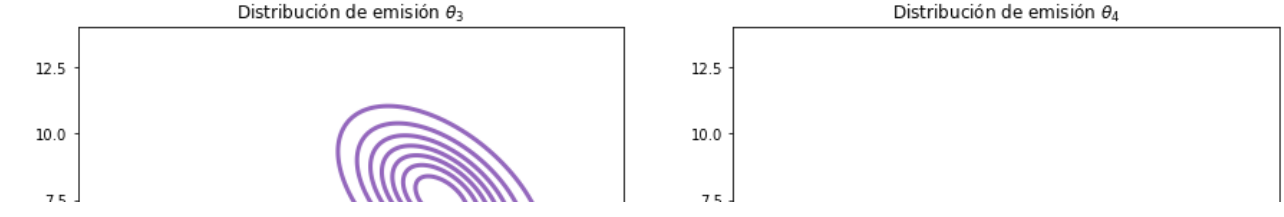
¿Qué tipo de HMM es según su topología?¿En qué difieren los modelos `HMM1` y `HMM2` ?

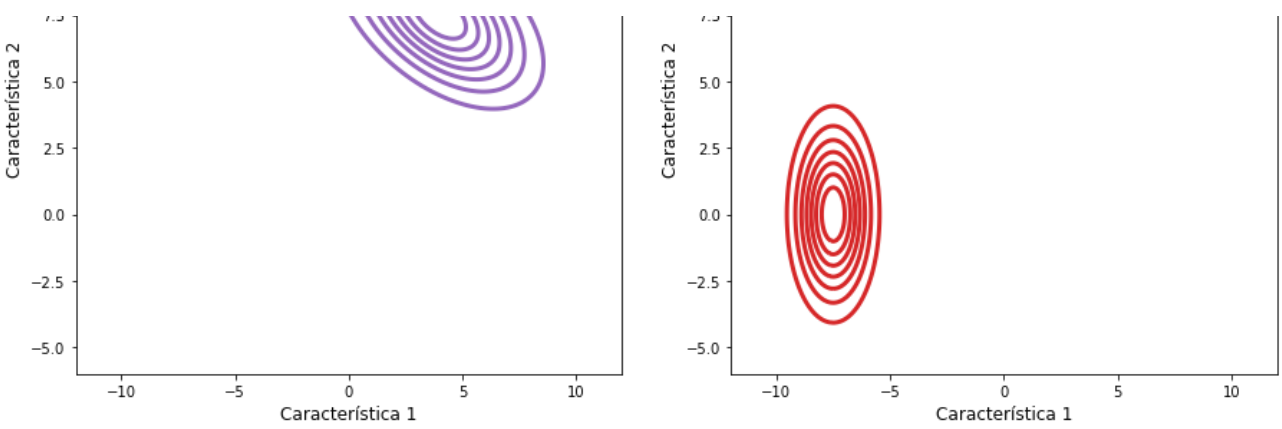
In [8]:

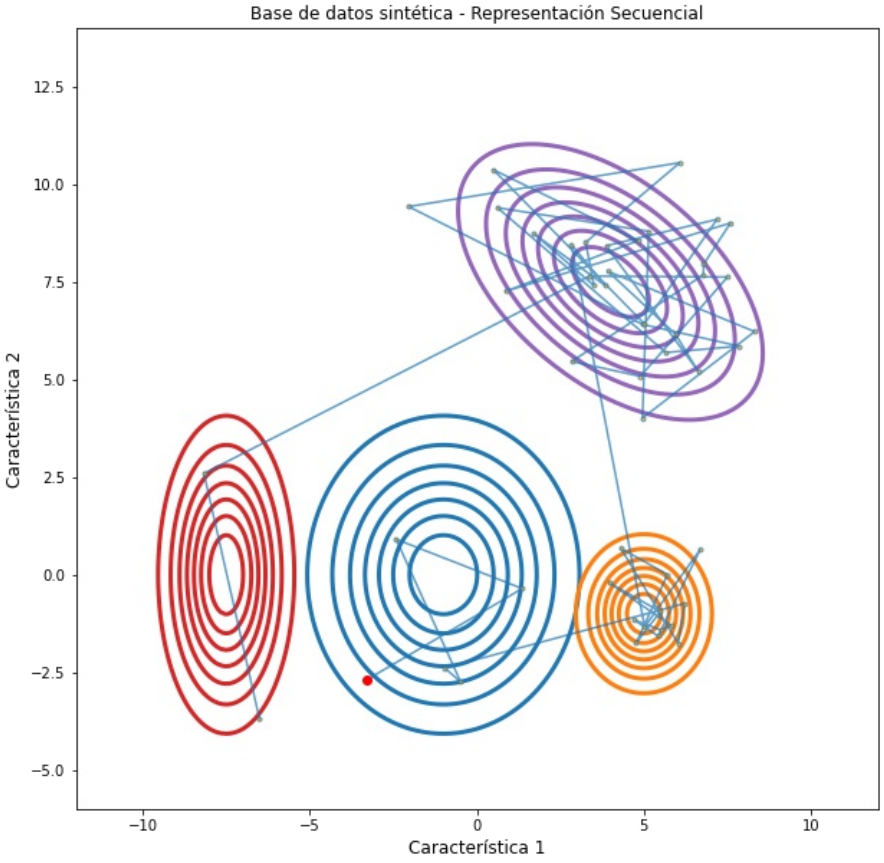
```
# Guardamos el modelo anterior con el nombre HMM1
import copy


# TO DO: Definición Matriz de Probabilidad de Transicion
A3 = np.array(
    [
        [0.7, 0.15, 0.15, 0.0],
        [0.0, 0.8, 0.15, 0.05],
        [0.0, 0.0, 0.9, 0.1],
        [0.0, 0.0, 0.0, 1.0],
    ]
)
# TO DO: Definición
pi3 = np.array([0.9, 0.1, 0.0, 0.0])

# TO DO: Definición del modelo HMM3
```

```
HMM3 = hmm.GaussianHMM(n_components=N, covariance_type="full")
HMM3.startprob_ = pi3
HMM3.transmat_ = A3
HMM3.means_ = mu
HMM3.covars_ = Sigma
HMM3.n_features = mu.shape[1]


# Apartado c)

# Generación de la figura
fig, ax = plt.subplots(2, 2, figsize=(15, 15))
ax = ax.ravel()

for n in range(N):

    # TO DO: Evaluación de cada una de las componentes gaussianas en los ejes
    eval_multivariate_gaussian = multivariate_normal.pdf(
        np.dstack((_X, _Y)), mean=HMM3.means_[n], cov=HMM3.covars_[n]
    )
    prob = np.reshape(eval_multivariate_gaussian, _X.shape)
    # Representación del contorno
    ax[n].contour(x_axis, y_axis, prob, colors=colors[n], linewidths=3)

    # Etiquetas y título
    ax[n].set_xlabel("Característica 1", fontsize=12)
    ax[n].set_ylabel("Característica 2", fontsize=12)
    ax[n].set_title(r"Distribución de emisión $\theta_{:.0f}$".format(n + 1))

# Apartado d)
# Muestreo ancestral -> Atributo .sample
# X: np.array NxD -> Secuencia de observaciones
# Z: np.array Nx1 -> Estados ocultos
X3, Z3 = HMM3.sample(T)


# Apartado e)
# Representación de los datos mediante un scatter plot
fig, ax = plt.subplots(figsize=(10, 10))

# Representación de los datos mediante un scatter plot
# Para mostrar el desarrollo de la secuencia
ax.plot(
    X3[:T_repr, 0], X3[:T_repr, 1], ".-", label="observations", mfc="orange", alpha=0.7
)
ax.plot(X3[0:1, 0], X3[0:1, 1], "o-r")
# plt.xlim((np.min(X3[:,0]),np.max(X3[:,0])))
# plt.ylim((np.min(X3[:,1]),np.max(X3[:,1])))
ax.set_xlabel("Característica 1", fontsize=12)
ax.set_ylabel("Característica 2", fontsize=12)
ax.set_title("Base de datos sintética - Representación Secuencial")

for n in range(N):
    # Evaluación de cada una de las componentes gaussianas en los ejes
    eval_multivariate_gaussian = multivariate_normal.pdf(
        np.dstack((_X, _Y)), mean=HMM3.means_[n], cov=HMM3.covars_[n]
    )
    prob = np.reshape(eval_multivariate_gaussian, _X.shape)
    # Representación del contorno
    ax.contour(x axis, y axis, prob, colors=colors[n], linewidths=3)
```









Parte 2: Problemas clásicos de los modelos ocultos de Markov.

2.1: Problema de puntuación

Dado el modelo $\lambda = (A, \mathbf{B}, \pi)$ y un conjunto de observaciones $\mathrm{X} = \{\mathrm{x}_1, \ldots, \mathrm{x}_N\}$ el cálculo de $P(X|\lambda)$ o la verosimilitud del modelo λ con los datos X se realiza mediante el algoritmo *Forward*.

Este algoritmo queda descrito por el siguiente pseudocodigo:

Algoritmo Forward Entrada: $\boldsymbol{X}, A, \mathbf{B}, \pi$

- $[\phi_1] = \mathrm{Eval\_Px\_z}(X_1, \mathbf{B})$
- $[\alpha_1, C_1] = \mathrm{Normalize}(\phi_1 \odot \pi)$
- **for** $t = 2 : N$
  do:
  - $[\phi_t] = \mathrm{Eval\_Px\_z}(X_t, \mathbf{B})$
  - $[\alpha_t, C_t] = \mathrm{Normalize}(\phi_t \odot (A^T \alpha_{t-1}))$
- **Return** $\alpha_{1:N}$
  , $\log P(X_{1:N}) = \sum_t \log C_t$

In [9]:

```
##############################################################################
# Función Normalize
# Descripción: Función para la normalización de un vector. Se va a utilizar para el calculo
# de las probabilidades a posteriori del estado P(Z|X) = P(X,Z)/P(X)
#
# Entradas:
#   u: np.array Kx1 correspondiente a P(X|Z)
```

```
# Salidas:
#   C: 1x1 P(X) probabilidad marginal de la secuencia de observaciones (marginalizando los estados)
#   alfa: Kx1 P(Z|X) probabilidades a posteriori de cada estado
################################################################################
def Normalize(u):
    C = np.sum(u)
    alfa = u / C

    return C, alfa
```

a.) Complete la función Eval_Px_z para calcular la probabilidad de observación dado cada un estado para el modelo `HMM1` en el que la probabilidad de observación de cada estado viene definida por una distribución Gaussiana bivariada.

Compruebe su funcionamiento con la primera muestra de nuestra base de datos X

In [10]:

```
################################################################################
# Función eval_Px_Z
# Descripción: Función para la obtención de P(X|Z). Evaluación de X en cada uno de
# las gaussianas (estados)
#
# Entradas:
#   data: np.array NxK secuencia de observaciones de entrada
#   B: Lista con las Gaussianas de cada estado. Lista de elementos clase multivariate_normal
# Salidas:
#    result: P(X|Z) : Probabilidad devuelta por el Gaussiana de cada estado
################################################################################
def eval_Px_Z(data, B):

    fi = np.empty((len(B), 1))
    for k, g in enumerate(B):
        # TO DO: Evaluación del la probabilidad de observación Gaussiana para la muestra
        fi[k] = g.pdf(data)
        # Se agrupan en una lista todas las probabilidades devueltas por cada estado
        # Se calcula la exponencial para obtener un valor de probabilidad.
    return fi
```

In [11]:

```
# Se almacenan las Gaussianas de cada estado en una lista.
# Al no ser entrenadas, necesitan la definición de todos sus parámetros.
B = []
for n in range(N):
    Gaussian_states = multivariate_normal(mean=mu[n], cov=Sigma[n])
    B.append(Gaussian_states)

# TO DO: LLamada a la función eval_Px_Z
fi = eval_Px_Z(X[0], B)

# TO DO: Devolver X[0], P(X[0]|lambda)
C, alpha = Normalize(HMM.startprob_.reshape((-1, 1)) * fi)

print("X[0]: ", X[0])
print("P(X[0]|lambda):")
print(alpha)
```

```
X[0]:  [ 4.4373229  -2.01628071]
P(X[0]|lambda):
[[9.68322546e-03]
 [9.90316774e-01]
 [9.34916978e-10]
 [2.22095354e-32]]
```

b.) A partir del pseudocódigo del enunciado, complete la función Forward que realiza el proceso completo para el cálculo de la verosimilitud de una secuencia
X **dado el HMM del primer ejercicio.**

Analice la función Forward ¿Qué representan los parámetros
ϕ,
α **y**
C**?**

In [12]:

```
################################################################################
# Función Forward
# Descripción: Implementa el algoritmo forward para la obtención de P(X|HMM)
#
# Entradas:
#   data: np.array NxK secuencia de observaciones de entrada
#   A:    np.array KxK Matriz de probabilidades de transición
#   pi:   np.list Kx1 Probabilidades de estado iniciales
#   B: Lista con los parámetros de los GMMs de cada estado
# Salidas:
#    log_likelihood: Logaritmo de la verosimilitud P(X\HMM)
################################################################################
def Forward(data, A, pi, B):
    # Inicialización de las variables: logaritmo de la verosimilitd
    # alpha: Matriz en la que se van a almacenar las P(Z|X)
    log_likelihood = 0
    alpha = np.zeros((len(B), np.size(data, 0)))

    # 1) Primera muestra
```

```
    pi = pi.reshape((len(B), 1))

    # TODO Calcular fi P(X_i|Z) sin normalizar
    fi = eval_Px_Z(data[0, :], B)

    # TODO calcular alfa P(X_i|Z) y C P(X_i)
    C, alfa = Normalize(fi * pi)

    # Almacenamiento de las prob
    alpha[:, 0] = alfa.ravel()

    # TO DO: Calculo del logaritmo de la probabilidad para la primera muestra
    log_likelihood = np.log(C)

    # 2) Para el resto de las muestras
    for i in range(1, np.size(data, 0)):

        # 2.1 TO DO:  Calculo de fi
        fi = eval_Px_Z(data[i, :], B)
        # 2.2 TO DO:calculo de alfa en el instante t dado alfa en t-1
        C, alfa = Normalize(fi * A.T @ alpha[:, i - 1])

        alpha[:, i] = alfa.ravel()
        # 2.3 TO DO: Se suma el logaritmo de la probabilidad marginal
        log_likelihood += np.log(C)

    return log likelihood
```

In [13]:

```
# LLamada a la función Forward -> Devuelve el logaritmo de la verosimilitud
loglikelihood = Forward(data=X, A=A, pi=pi, B=B)
print(
    "El logaritmo de la verosimilitud del modelo con la secuencia de observaciones X P(X|\u03BB) es {:.4f}".format(
        loglikelihood
    )
)
```

```
El logaritmo de la verosimilitud del modelo con la secuencia de observaciones X P(X|λ) es -17683.6125
```

b.) Compruebe el resultado obtenido en el apartado anterior con el resultado al aplicar la función de `hmmlearn` (método `.score`) para el cálculo de la verosimilitud mediante el algoritmo *Forward*.

In [14]:

```
# TO DO: Implementación del algoritmo Forward en el atributo .score del HMM
loglikelihood = HMM.score(X)

print(
    "El logaritmo de la verosimilitud del modelo con la secuencia de observaciones X P(X|\u03BB) es {:.4f}".format(
        loglikelihood
    )
)
```

```
El logaritmo de la verosimilitud del modelo con la secuencia de observaciones X P(X|λ) es -17683.6125
```

Vemos que los resultados son idénticos a los que nos ofrece el método `score`.

c.) ¿Cuál de los tres modelos `HMM`, `HMM2` y `HMM3` maximiza el logaritmo de la verosimilitud? ¿Es este resultado el esperado? Justifique su respuesta.

`Nota`: Utilice para ello la función `Forward` implementada anteriormente o la función disponible en `hmmlearn`.

In [15]:

```
loglikelihood = HMM2.score(X)

print(
    "El logaritmo de la verosimilitud del modelo con la secuencia de observaciones X P(X|\u03BB) es {:.4f}".format(
        loglikelihood
    )
)
```

```
El logaritmo de la verosimilitud del modelo con la secuencia de observaciones X P(X|λ) es -19564.3238
```

In [16]:

```
loglikelihood = Forward(data=X, A=A3, pi=pi3, B=B)
print(
    "El logaritmo de la verosimilitud del modelo con la secuencia de observaciones X P(X|\u03BB) es {:.4f}".format(
        loglikelihood
    )
)
```

```
El logaritmo de la verosimilitud del modelo con la secuencia de observaciones X P(X|λ) es -37091.7911
```

In [17]:

```
loglikelihood = HMM3.score(X)

print(
    "El logaritmo de la verosimilitud del modelo con la secuencia de observaciones X P(X|\u03BB) es {:.4f}".format(
        loglikelihood
    )
)
```

```
El logaritmo de la verosimilitud del modelo con la secuencia de observaciones X P(X|λ) es -37091.7911
```

2.1: Problema de reconocimiento de estados

Dado una secuencia de observaciones $X = \{x_1, \ldots, x_N\}$
y un modelo $\lambda = (\pi, A, \mathbf{B})$
el algoritmo de *viterbi* permite encontrar la secuencia de estados ocultos más probable , es decir, permite obtener la secuencia de estados que mejor ''explica'' las observaciones.

El algoritmo de viterbi queda descrito por el siguiente pseudocódigo:

Algoritmo de Viterbi **Entrada:** X
, A
, π
, $\mathbf{B}$

- $a_1 = \mathbf{0}$
- $\phi_1 = \text{eval\_Px\_Z}(X_1, \mathbf{B})$
- $\delta = \pi \odot \phi_1$
- **for** $t = 2 : T$
  do:
  - **for** $j = 1 : N$
    do:
  - $\phi_t = \text{eval\_Px\_Z}(X_t, \mathbf{B})$
  - $[a_t(j),\ \delta_t(j)] = \max_j(\log\delta_{t-1}(:) + \log A_{ij} + \log\phi_t(j))$
- $S_T = \text{argmax}(\delta_T)$
- **for** $t = N - 1 : 1$
  do
  - $S_t = a_{t-1} S_{t+1}$
- **Return S**

a.) Complete la función `viterbi` para que integre el algoritmo de viterbi para la decodificación de la secuencia más probable de estados dada una secuencia de observaciones.

Describa brevemente el significado de cada una de las variables del algoritmo y compruebe la secuencia resultante con los estados obtenidos al generar la base de datos.

In [18]:

```
# %% Implementación del algoritmo de Viterbi:
import warnings

warnings.filterwarnings("ignore")

###############################################################################
# Función Viterbi
# Descripción: Implementa el algoritmo de viterbi para la generación de la secuencia
# de estados más probable dado un conjunto de observaciones y un modelo
#
# Entradas:
#   data: np.array NxK secuencia de observaciones de entrada
#   A:    np.array KxK Matriz de probabilidades de transición
#   pi:   np.list Kx1 Probabilidades de estado iniciales
#   B: Lista con los parámetros de los GMMs de cada estado
# Salidas:
#    S:  np.array Nx1 con la secuencia de estados más probable
###############################################################################
def viterbi(data, A, B, pi):
    # Inicialización de Matriz en la que vamos a ir guardando los estados más probables
    a = np.zeros((np.size(A, 0), np.size(data, 0) - 1))
    # Inicialización en la que se van a ir guardando las mejores puntuaciones
    omega = np.zeros((np.size(A, 0), np.size(data, 0)))

    # 1) Incializacion Primera muestra
    pi = pi.reshape((len(B), 1))
    fi = eval_Px_Z(data[0, :], B)
    omega[:, 0] = np.log((pi * fi).ravel())

    # 2) Recursión
    for t in range(1, np.size(data, 0)):
        for k in range(len(B)):

            # TO DO: Calculo de P(X|Z)
            fi = eval_Px_Z(data[t, :], B)

            # TO DO: calculo del logaritmo de las puntuaciones
            prob = omega[:, t - 1] + np.log(A[:, k]) + np.log(fi[k])

            # TO DO: Calculo de la probabilidad máxima
            omega[k, t] = np.max(prob)

            # TO DO: Calculo del estado anterior más probable
            a[k, t - 1] = np.argmax(prob)

    # Array con la secuencia de estados final
```

```
    S = np.zeros(np.size(data, 0))

    # 3) Terminación
    ultimo_estado = np.argmax(omega[:, -1:])
    S[0] = ultimo_estado

    # 4) Path Backtracking
    indice_back = 1
    for i in range(np.size(data, 0) - 2, -1, -1):
        S[indice_back] = a[int(ultimo_estado), i]
        ultimo_estado = S[indice_back]
        indice_back += 1

    S = np.flip(S, axis=0)
    return S
```

In [19]:

```
# TO DO: LLamada a viterbi
SeqViterbi = viterbi(X, A, B, pi)
print("La secuencia de estados ocultos más probable es ", SeqViterbi)
```

```
La secuencia de estados ocultos más probable es  [1. 1. 1. ... 2. 2. 2.]
```

b.) Haga uso de la función `.decode` de `hmmlearn` para la implementación del algoritmo viterbi para la decodificación de la secuencia más probable de estados dada una secuencia de observaciones. ¿Coincide la secuencia con la devuelta por la función previamente implementada?

In [20]:

```
# TO DO: Implementacion de viterbi mediante .decode(algorithm='viterbi')
HMM.decode(X, algorithm="viterbi")
```

Out[20]:

```
(-17721.908136109232, array([1, 1, 1, ..., 2, 2, 2]))
```

Comprobamos que hemos obtenido la misma secuencia.

In [21]:

```
(HMM.decode(X, algorithm="viterbi")[1] == SeqViterbi).all()
```

Out[21]:

```
True
```

2.3: Problema de entrenamiento

Dado un conjunto de observaciones $X = \{x_1, \ldots, x_N\}$
el cálculo de los parámetros que definen el modelo oculto de markov $\lambda = (\pi, A, B)$
que maximizan la verosimilitud de los datos dado el modelo $P(X|\lambda)$
se obtiene mediante el algoritmo EM o, en este caso, el algoritmo *Baum-Welch*.

a) Utilice la función `.fit` para entrenar un modelo oculto de markov a partir de los datos sintéticos generados en el ejercicio 1.

Analice los parámetros obtenidos y compárelos con el modelo generador de los datos. ¿Qué ocurre con el parámetro de los estados iniciales?

In [22]:

```
# TO DO: Utilización de .fit para el entrenamiento de la clase Gaussian HMM
model_trained = HMM.fit(X)

print("Mean:\n", np.round(HMM.means_, 3))
print("Sigma:\n", np.round(HMM.covars_, 3))
print("pi:\n", np.round(HMM.startprob_, 3))
```

```
Even though the 'startprob_' attribute is set, it will be overwritten during initialization because 'init_params' conta
ins 's'
Even though the 'transmat_' attribute is set, it will be overwritten during initialization because 'init_params' contai
ns 't'
Even though the 'means_' attribute is set, it will be overwritten during initialization because 'init_params' contains
'm'
Even though the 'covars_' attribute is set, it will be overwritten during initialization because 'init_params' contains
'c'
```

```
Mean:
 [[-0.796  0.011]
 [ 3.951  7.608]
 [-7.452  0.025]
 [ 5.042 -1.011]]
Sigma:
 [[[ 4.357e+00  2.120e-01]
  [ 2.120e-01  3.930e+00]]

 [[ 5.219e+00 -2.140e+00]
  [-2.140e+00  3.109e+00]]

 [[ 9.380e-01  1.200e-02]
  [ 1.200e-02  3.682e+00]]

 [[ 1.005e+00  1.000e-03]
  [ 1.000e-03  9.890e-01]]]
pi:
 [0. 0. 0. 1.]
```

b.) Realice el entrenamiento del modelo utilizando varias secuencias generadas sintéticamente a partir del muestreo ancestral del primer modelo. Compare los parámetros del modelo aprendido con el primer modelo.

In [23]:

```
# Supongamos que tenemos diferentes secuencias de entrenamiento
X1, Z1 = HMM.sample(4000)
X2, Z2 = HMM.sample(4000)
X3, Z3 = HMM.sample(4000)
X4, Z4 = HMM.sample(4000)
X5, Z5 = HMM.sample(4000)

# Concatenamos todas las secuencias que tenemos
X_total = np.concatenate([X, X1, X2, X3, X4, X5], axis=0)
# Determinamos la longitud de cada cadena
lengths = np.array([len(X), len(X1), len(X2), len(X3), len(X4), len(X5)])

# TO DO: Utilización de .fit para el entrenamiento de la clase Gaussian HMM
model_trained = HMM.fit(X_total, lengths=lengths)

print("Mean:\n", np.round(HMM.means_, 3))
print("Sigma:\n", np.round(HMM.covars_, 3))
print("pi:\n", np.round(HMM.startprob_, 3))
```

```
Even though the 'startprob_' attribute is set, it will be overwritten during initialization because 'init_params' conta
ins 's'
Even though the 'transmat_' attribute is set, it will be overwritten during initialization because 'init_params' contai
ns 't'
Even though the 'means_' attribute is set, it will be overwritten during initialization because 'init_params' contains
'm'
Even though the 'covars_' attribute is set, it will be overwritten during initialization because 'init_params' contains
'c'
```

```
Mean:
 [[-7.441e+00  4.000e-03]
 [ 5.059e+00 -1.028e+00]
 [ 3.929e+00  7.633e+00]
 [-7.600e-01 -2.300e-02]]
Sigma:
 [[[ 0.938 -0.022]
  [-0.022  3.765]]

 [[ 1.017 -0.019]
  [-0.019  1.007]]

 [[ 5.213 -2.175]
  [-2.175  3.145]]

 [[ 4.532  0.076]
  [ 0.076  3.953]]]
pi:
 [0. 1. 0. 0.]
```

Parte 3: Clasificación de patrones temporales mediante HMMs

Caso Ficticio: Una empresa dedicada al suministro y gestión de maquinaria industrial desea identificar si las averías presentes en sus equipos se deben a un fallo en un componente o debido a un uso indebido de los mismos. Esto es considerado una tarea clave para la empresa, pues en el caso de que las averías se debiesen a un fallo debido a su incorrecto uso, serían los usuarios los responsables de financiar el arreglo.

Para ello, el personal técnico decide instalar determinados sensores para medir la vibración y la velocidad de funcionamiento de las máquinas. Tras varios meses de funcionamiento, los sensores capturan secuencias de ambas medidas normalizadas en diferentes circunstancias.

De esta forma, se recogen 10 secuencias asociadas a averías debido al fallo de un componente, 10 secuencias asociadas a averías debido a su uso indebido y, por último, 10 secuencias en las que el equipo no presenta avería alguna.

Las medidas se pueden encontrar en los ficheros "uso_inapropiado.csv", "fallo_componente.csv" y "sin_averias.csv".

De cara a poder clasificar dichas averías, se decide entrenar un modelo oculto de Markov para modelar cada una de los casos de estudio.

a.) Cargue los datos de entrenamiento haciendo uso de la función `pandas.read_csv`.

In [24]:

```
# Carga de los datos
import pandas as pd

# uploaded = files.upload()
# uploaded = files.upload()
# uploaded = files.upload()


sin_averias = pd.read_csv("../../../Data/Datos2/sin_averias.csv", index_col=0).values
fallo_componente = pd.read_csv("../../../Data/Datos2/fallo_componente.csv", index_col=0).values
uso_inapropiado = pd.read_csv("../../../Data/Datos2/uso_inapropiado.csv", index_col=0).values
```

b.) Realice el entrenamiento de un modelo oculto de markov con distribuciones de emisión gaussianas que modele cada uno de los casos de estudio. Para ello, al igual que se hizo en el ejercicio anterior, utilice el paquete `hmmlearn`.

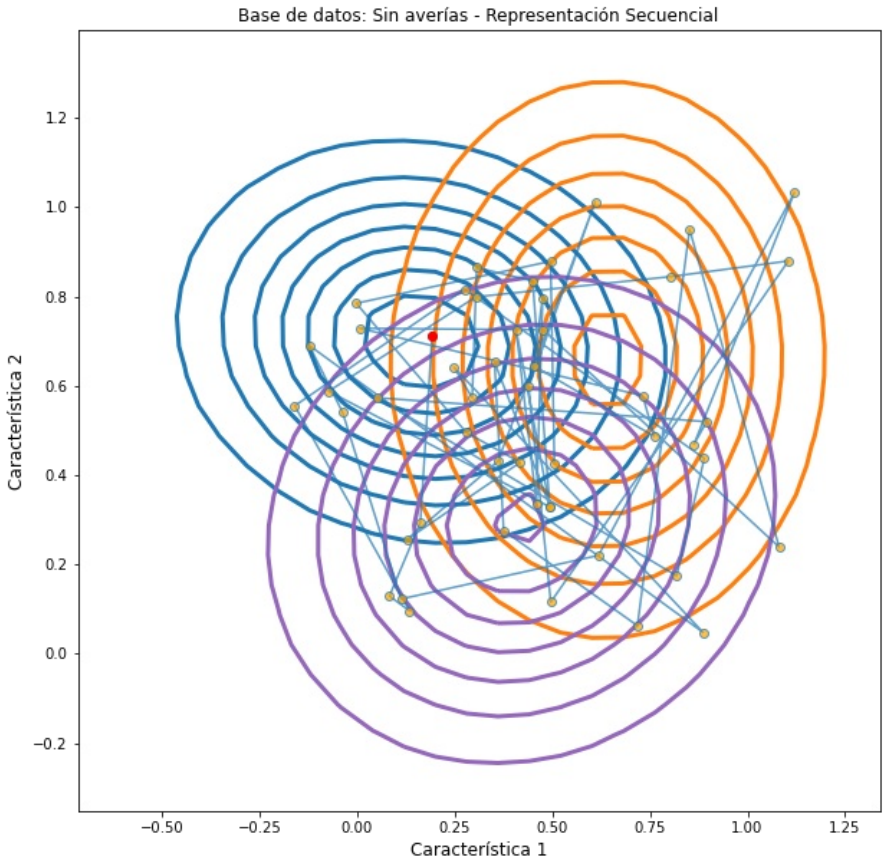
Represente los parámetros de cada uno de los modelos.

In [25]:

```
# Entrenamiento de un modelo por cada clase
lengths = np.array([50 for i in range(10)])
```

In [26]:

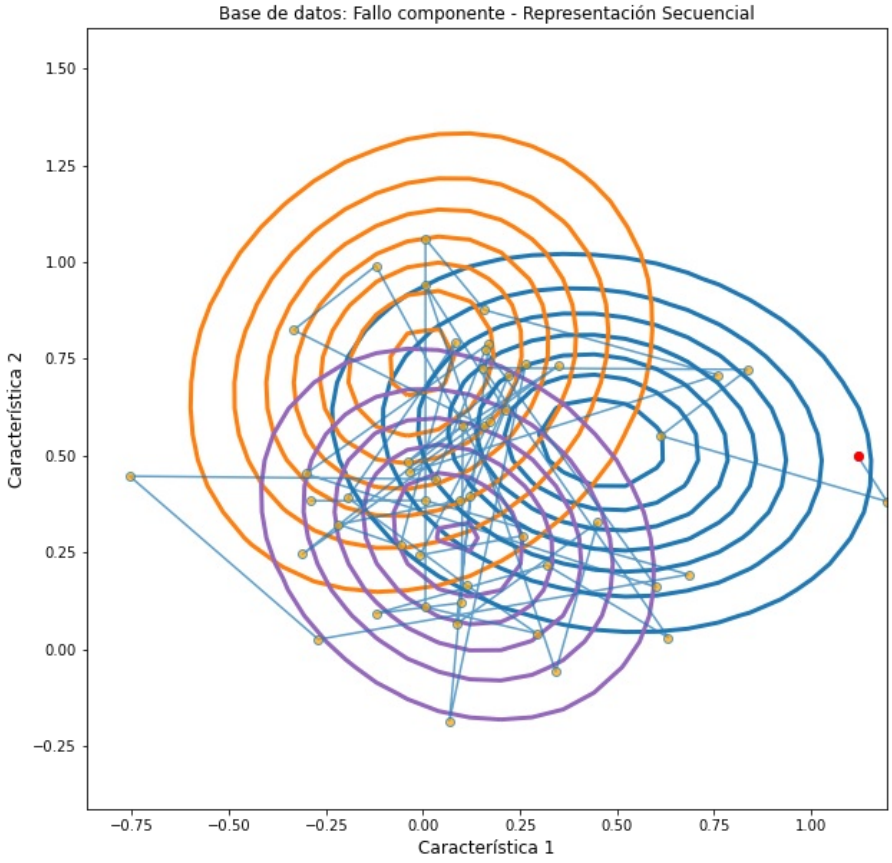
```
N = 3

HMM_sin_averias = hmm.GaussianHMM(
    n_components=N, covariance_type="full", init_params=""
)
HMM_sin_averias.fit(sin_averias, lengths)

T_repr = 50

# Representación de los datos mediante un scatter plot
fig, ax = plt.subplots(figsize=(10, 10))

# Representación de los datos mediante un scatter plot
# Para mostrar el desarrollo de la secuencia
ax.plot(
    sin_averias[:T_repr, 0],
    sin_averias[:T_repr, 1],
    "o-",
    label="observations",
    mfc="orange",
    alpha=0.7,
)
ax.plot(sin_averias[0:1, 0], sin_averias[0:1, 1], "or")
plt.xlim((np.min(sin_averias[:, 0]), np.max(sin_averias[:, 0])))
plt.ylim((np.min(sin_averias[:, 1]), np.max(sin_averias[:, 1])))
ax.set_xlabel("Característica 1", fontsize=12)
ax.set_ylabel("Característica 2", fontsize=12)
ax.set_title("Base de datos: Sin averías - Representación Secuencial")

for n in range(N):
    #  TO DO: Evaluación de cada una de las componentes gaussianas en los ejes
    # y representación de los contornos
    eval_multivariate_gaussian = multivariate_normal.pdf(
        positions, mean=HMM_sin_averias.means_[n], cov=HMM_sin_averias.covars_[n]
    )
    prob = np.reshape(eval_multivariate_gaussian, _X.shape)
    # Representación del contorno
    ax.contour(x_axis, y_axis, prob, colors=colors[n], linewidths=3)

plt.show();
```



In [27]:

```
N = 3

HMM_fallo_componente = hmm.GaussianHMM(
    n_components=N, covariance_type="full", init_params=""
)
HMM_fallo_componente.fit(fallo_componente, lengths)


T_repr = 50
```

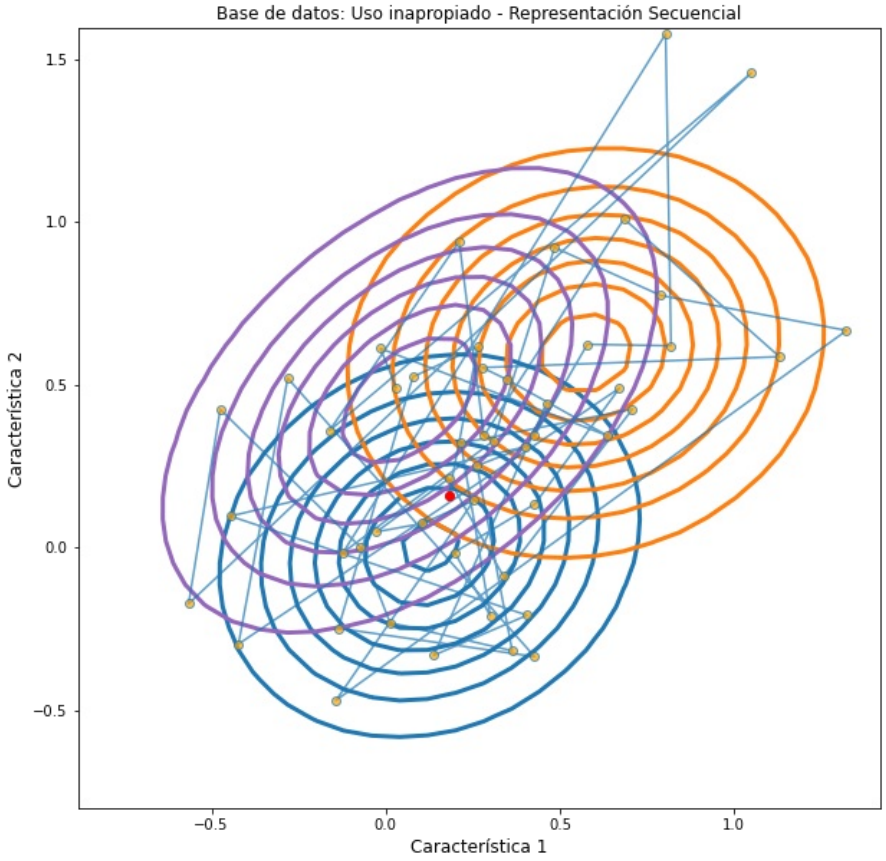
```
# Representación de los datos mediante un scatter plot
fig, ax = plt.subplots(figsize=(10, 10))

# Representación de los datos mediante un scatter plot
# Para mostrar el desarrollo de la secuencia
ax.plot(
    fallo_componente[:T_repr, 0],
    fallo_componente[:T_repr, 1],
    "o-",
    label="observations",
    mfc="orange",
    alpha=0.7,
)
ax.plot(fallo_componente[0:1, 0], fallo_componente[0:1, 1], "or")
plt.xlim((np.min(fallo_componente[:, 0]), np.max(fallo_componente[:, 0])))
plt.ylim((np.min(fallo_componente[:, 1]), np.max(fallo_componente[:, 1])))
ax.set_xlabel("Característica 1", fontsize=12)
ax.set_ylabel("Característica 2", fontsize=12)
ax.set_title("Base de datos: Fallo componente - Representación Secuencial")

for n in range(N):
    #  TO DO: Evaluación de cada una de las componentes gaussianas en los ejes
    # y representación de los contornos
    eval_multivariate_gaussian = multivariate_normal.pdf(
        positions,
        mean=HMM_fallo_componente.means_[n],
        cov=HMM_fallo_componente.covars_[n],
    )
    prob = np.reshape(eval_multivariate_gaussian, _X.shape)
    # Representación del contorno
    ax.contour(x_axis, y_axis, prob, colors=colors[n], linewidths=3)

plt.show();
```



In [28]:

```
N = 3

HMM_uso_inapropiado = hmm.GaussianHMM(
    n_components=N, covariance_type="full", init_params=""
)
HMM_uso_inapropiado.fit(uso_inapropiado, lengths)


T_repr = 50

# Representación de los datos mediante un scatter plot
fig, ax = plt.subplots(figsize=(10, 10))

# Representación de los datos mediante un scatter plot
# Para mostrar el desarrollo de la secuencia
ax.plot(
    uso_inapropiado[:T_repr, 0],
    uso_inapropiado[:T_repr, 1],
    "o-",
    label="observations",
    mfc="orange",
    alpha=0.7,
```

```
)
ax.plot(uso_inapropiado[0:1, 0], uso_inapropiado[0:1, 1], "or")
plt.xlim((np.min(uso_inapropiado[:, 0]), np.max(uso_inapropiado[:, 0])))
plt.ylim((np.min(uso_inapropiado[:, 1]), np.max(uso_inapropiado[:, 1])))
ax.set_xlabel("Característica 1", fontsize=12)
ax.set_ylabel("Característica 2", fontsize=12)
ax.set_title("Base de datos: Uso inapropiado - Representación Secuencial")

for n in range(N):
    #  TO DO: Evaluación de cada una de las componentes gaussianas en los ejes
    # y representación de los contornos
    eval_multivariate_gaussian = multivariate_normal.pdf(
        positions,
        mean=HMM_uso_inapropiado.means_[n],
        cov=HMM_uso_inapropiado.covars_[n],
    )
    prob = np.reshape(eval_multivariate_gaussian, _X.shape)
    # Representación del contorno
    ax.contour(x_axis, y_axis, prob, colors=colors[n], linewidths=3)

plt.show();
```



c.) Tras el entrenamiento de los modelos, se obtiene una secuencia relacionada con la posible avería de un equipo nuevo. Cargue los datos del fichero `test.csv`.

In [29]:

```
test sequence = pd.read csv("../../../Data/Datos2/test.csv", index col=0).values
```

A partir de los modelos generados anteriormente ¿Está la máquina averiada?

En caso de estar averiada ¿Dicha avería es debido a un fallo de fabricación o a un uso indebido?

In [30]:

```
log_likelihood_usoInapropiado = HMM_uso_inapropiado.score(test_sequence)
log_likelihood_falloComponente = HMM_fallo_componente.score(test_sequence)
log_likelihood_sinAverias = HMM_sin_averias.score(test_sequence)


print("Log-likelihood Uso Inapropiado:", log_likelihood_usoInapropiado)
print("Log-likelihood Uso Fallo Componente:", log_likelihood_falloComponente)
print("Log-likelihood Uso Sin Averias:", log likelihood sinAverias)
```

```
Log-likelihood Uso Inapropiado: -93.54246592220697
Log-likelihood Uso Fallo Componente: -143.1429231705949
Log-likelihood Uso Sin Averias: -145.25171820265723
```

Vemos que el modelo más probable es el de de Uso Inapropiado, con una log-verosimilitud de -93.54 (el más próximo a 0), lo que se traduce en un uso inadecuado por parte de los empleados.